Identifying Fraud from Enron Emails and Financial Data

By Yongnan Sun, in fulfillment of Udacity's Data Analyst Nanodegree, Project 5

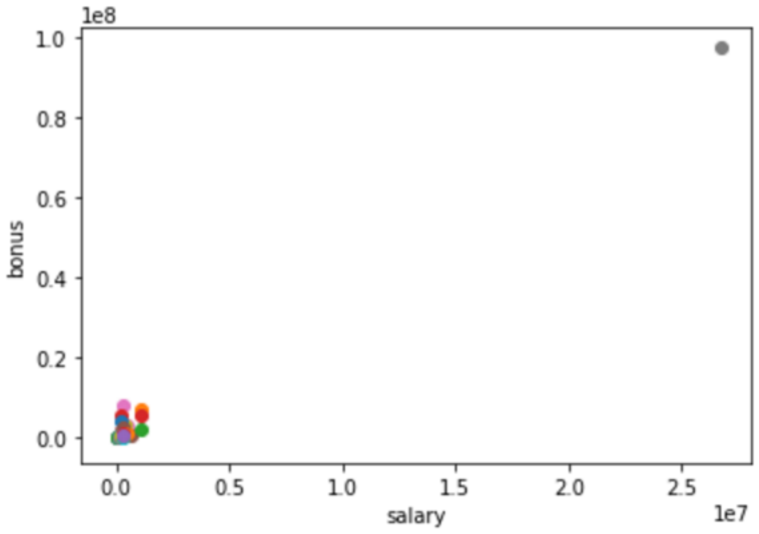
1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those? [relevant rubric items: “data exploration”, “outlier investigation”]

The Enron email corpus is a compilation of emails sent to and from important Enron employees during the period during which major financial fraud was being committed. By evaluating data from the Enron email corpus and public financial reports using machine learning techniques, we are trying to determine who within the Enron organization should be considered a “person of interest”. The dataset contains information about many potential POIs’ financial interest in Enron as well as their email activity to and from other Enron employees, including potential POIs.

The dataset contains a total of **146 data points** with **21 features**, **18 POIs** and **128 Non POIs**.

After Exploratory Data Analysis, I found three 3 records need removal:

- TOTAL: Contains extreme values for most numerical features(below)
- THE TRAVEL AGENCY IN THE PARK: Not represent an individual. Found by looking at the features by hand
- LOCKHART EUGENE E: Contains only NaN values



I removed the TOTAL because it was an extreme outlier. Somehow, the “Total” sum of all records was included in the dataset. A plot of bonus vs. salary shows this data point clearly. By looking at the persons names I found there was a strange name ‘THE TRAVEL AGENCY IN THE PARK’, which did not represent an individual so I decided to remove it. There is another record pertaining to ‘LOCKHART EUGENE E’ that has no numbers for financial data. I didn’t need to write any code to address this issue, because featureFormat from tester.py takes care of that case.

The machine learning techniques involve picking an appropriate set of features, scaling those features, potentially reducing dimensionality, applying a classification algorithm, and evaluating the results on multiple dimensions.

2. What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values. [relevant rubric items: “create new features”, “intelligently select features”, “properly scale features”]

The features (with SelectKBest feature scores) I ended up using are the following:

- Salary
- Bonus
- deferred_income
- total_stock_value
- exercised_stock_options
- fraction_to_poi

The model I ended up using is DecisionTree so I did not do scaling. While for other algorithms like SVM I implemented feature scaling like MinMaxScaler and StandardScaler. I defined a function called ‘get_selected_features_scores_pvalues()’ which takes a clf as input and print the 'Selected Features, Scores, P-Values'. By using this function I got the result:

Selected Features, Scores, P-Values:

[('exercised_stock_options', '24.82', '0.000'), ('total_stock_value', '24.18', '0.000'), ('bonus', '20.79', '0.000'), ('salary', '18.29', '0.000'), ('fraction_to_poi', '16.58', '0.000'), ('deferred_income', '11.46', '0.001')].

I selected those features using SelectKBest as the first step in my pipeline. A very important parameter of SelectKBest in this step is “k”, the number of parameters that the algorithm is allowed to select. I started my evaluation with the following code in my “params” dictionary of pipeline parameters:

"skb__k": [6, 7, 8, 9, 10, 11, 12]

I made three of my own features:

- fraction_from_poi: The percentage of all emails to a person that came from a POI. The rationale is to see if people who receive a high percentage of their emails from POIs are in fact themselves POIs.
- fraction_to_poi: The percentage of all emails that a person sent that were sent to a POI. I added this to see if a person who sends a high percentage of their emails to POIs is himself or herself a POI.
- fraction_salary_total_payments: The percentage of a person's take-home pay that was fixed as opposed to a bonus or stock grant. My guess was that the more variable a person's compensation was, the more likely it would be that they would commit fraud.

It turns out that the second feature I created - 'fraction_to_poi' was one of the features that I used in my final classification algorithm.

3. What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: "pick an algorithm"]

   I ended up using the DecisionTree algorithm. I also tried Naïve Bayes, KNN, Random Forest and SVM.

ALGORITHM	ACCURACY	PRECISION	RECALL
Naïve Bayes	0.860	0.464	0.312
KNN	0.833	0.314	0.215
DecisionTree	**0.848**	**0.417**	**0.360**
Random Forest	0.851	0.391	0.206
SVM	0.863	0.449	0.127
Average	0.851	0.407	0.244

4. What does it mean to tune the parameters of an algorithm, and what can happen if you don't do this well? How did you tune the parameters of your particular algorithm? What parameters did you tune? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric items: "discuss parameter tuning", "tune the algorithm"]

   Tuning an algorithm means finding the optimal combination of configuration parameters that yields the highest "score". In this case, we are looking for the highest F1 score, which combines precision and recall into one value that an algorithm can maximize. If you don't do this well, the classifier will not be optimized to be as predictive as it could be.

I used GridSearchCV, an automated way of running multiple iterations of the same algorithm using different parameter combinations in search of the one that yields the highest score. As mentioned above, in this case, the high score is based on the “F1,” During the GridSearchCV step, I used Stratified Shuffle Split on the labels in an effort to randomize the selection of testing data due to the small sample size.

I ultimately selected DecisionTree. Below is my parameters dictionary:

```
params = {"pca__n_components": [2, 3, 4, 5, 6],

        "skb__k": [6, 7, 8, 9, 10, 11, 12],

        #'tree__criterion': ['gini', 'entropy'],

        #'tree__splitter': ['best', 'random'],

        'tree__min_samples_split': [3, 4, 5, 6]}
```

I tried different combinations of tree__criterion, tree__splitter and tree__min_samples_split and got a relatively good F1 score for DecisionTree.

5. What is validation, and what’s a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric items: “discuss validation”, “validation strategy”]

Validation is performed to ensure that a machine learning algorithm generalizes well. Validation involves separating data into training and testing data sets, using the training set to train the algorithm, and then validating that the algorithm works against another set of data, the testing set. A classic mistake is over-fitting, where the model is trained and performs very well on the training dataset, but markedly worse on the test dataset.

To validate the performance of each algorithm, I used the test_classifier function in tester.py with the winning classifier from the best_estimator_ result. The test_classifier function uses Stratified Shuffle Split because the number of observations in our dataset is relatively small, and the nonPOIs outnumber the POIs 7 to 1. Going with Stratified K Folds would not have given us the ability to run a large number of randomized datasplitting scenarios (folds) that give us the best chance to analyze such a small dataset.

By comparing the results of test_classifier for the best_estimator_ of different algorithms, I chose the algorithm that gave me the best combination of precision and recall. In this case, as detailed in previous sections, DecisionTree turned out to be the winner.

6. Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-

understandable about your algorithm’s performance. [relevant rubric item: “usage of evaluation metrics”]

The main evaluation metrics utilized were precision and recall. When a classifier has a high level of precision , it yields a low rate of false positives, meaning when a POI is present in the data, we are good at flagging him or her. Across all of the algorithms I tried, the average is 0.407. My finding means that, when the classifier predicts a person is a POI, that person is actually a POI 41.7% of the time.

When a classifier has a high level of recall , it yields a low rate of false negatives, meaning there is less of a likelihood we are going to ignore a person of interest when we should really be be looking at him or her more closely. Across all of the algorithms I tried, the average is 0.244. My finding means that, my classifier identifies 36% of all POIs correctly as true POIs.